Task/Subtask ID52.1(2)
CDRL Sequence 05504-001
31 July 1993

SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS (STARS) PROGRAM

Cleanroom Engineering Handbook Volume 4 Specification Team Practices

Contract No. F19628-88-D-0032

Task ID52 – STARS Technology Transfer Demonstration Project for the U.S. Army

Prepared for:

Electronic Systems Center
Air Force Materiel Command, USAF
Hanscom AFB, MA 01731-2816

Prepared by:

IBM Federal Systems Company
800 North Frederick Avenue
Gaithersburg, MD 20879

Task/Subtask ID52.1(2)
CDRL Sequence 05504-001
31 July 1993

SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS (STARS) PROGRAM

Cleanroom Engineering Handbook Volume 4 Specification Team Practices

Contract No. F19628-88-D-0032

Task ID52 – STARS Technology Transfer Demonstration Project for the U.S. Army

Prepared for:

Electronic Systems Center
Air Force Materiel Command, USAF
Hanscom AFB, MA 01731-2816

Prepared by:

IBM Federal Systems Company
800 North Frederick Avenue
Gaithersburg, MD 20879

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0166

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE 7/31/93	3. REPORT TYPE AND DATES COVERED Initial

4. TITLE AND SUBTITLE

Cleanroom Engineering Handbook:
Specification Team Practices

5. FUNDING NUMBERS

F19628-88-C-0032/0010

6. AUTHOR(S)

Ara Kouchakdjian
Alan R. Hevner
Richard H. Cobb
James A. Whittaker

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

IBM Federal Systems Company
800 North Frederick Avenue
Gaithersburg, MD 20879

SET, Inc.
2770 Indian River Blvd.
Vero Beach, FL 32960

8. PERFORMING ORGANIZATION REPORT NUMBER

05504-001
Volume 4

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Electronic Systems Center/ENS
Air Force Materiel Command, USAF
5 Eglin Street, Building 1704
Hanscom Air Force Base, MA 01731-2116

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

N/A

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Cleared for Public Release, Distribution is Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This is one of a series of six engineering handbooks prepared for and used by the engineering staff at Picatinny Arsenal for the STARS technology transfer demonstration. The handbooks define the engineering process and algorithms that will be used in Cleanroom projects. They are designed to provide support to trained engineers using Cleanroom Engineering, not to substitute for training.

This handbook, Volume 4, explains the set of specific tasks performed by the Cleanroom Specification Team for each cycle of a project. Specification tasks are described and the relationship among tasks are defined. The Cleanroom specification team's mission is to create the system specification in six specification volumes. Templates are proposed as recommended formats for the presentation of information and tasks results. Techniques for revising the specifications throughout the system life cycle are described.

The Cleanroom process model for software system development projects is presented in Volume 1 - Cleanroom Process Overview - of this series of Cleanroom Handbooks. This handbook, Volume 4, describes the activities of the Specification Team for each cycle of project development. The specification activity may include the preparation of both system and software specifications, depending on the needs and status of the project. Specifications define the requirements placed on the system so it can satisfy its mission and the behavior the system must exhibit to fulfill the assigned requirements. A system specification defines requirements and behavior for a solution which integrates some combination of hardware, software, communi-cations, human behavior, etc. A software specification defines requirements and behavior for the software components of the system. The focus of this volume is on the preparation of software specifications. The techniques described are useful for preparing system specifications but in order to keep the handbook within a manageable focus, this presentation is limited to software.

14. SUBJECT TERMS

Certification, Cleanroom, Cleanroom Engineering, Development, Management, Software Development, Specification

15. NUMBER OF PAGES

69

16. PRICE CODE

N/A

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	SAR

PREFACE

This series of handbooks is prepared for use by managers and engineers assigned to Cleanroom projects at Picatinny Life Cycle Software Engineering Center.

These handbooks define the engineering process and algorithms that will be used in Cleanroom projects.

This document was developed by the IBM Federal Systems Company, located at 800 North Frederick Avenue, Gaithersburg, MD 20879 and Software Engineering Technology, Inc. located at 2770 Indian River Boulevard, Vero Beach, FL 32960. Questions or comments should be directed to Mr. Paul Arnold at 301-240-7464 (Internet: pga@sei.cmu.edu).

CLEANROOM ENGINEERING SPECIFICATION

TABLE OF CONTENTS

CLEANROOM ENGINEERING SPECIFICATION

SECTION 1: INTRODUCTION

The mission of this handbook for Cleanroom Specification is to organize and explain the set of specific tasks performed by the Cleanroom Specification Team for each cycle of a project. Specification tasks are described and the relationships among tasks are defined.

The Cleanroom Specification Team mission is to create the system specification in six specification volumes. Templates are proposed as recommended formats for the presentation of information and task results. Techniques for revising the specifications throughout the system life cycle are described.

The Cleanroom process model for software system development projects is presented in *Volume 1 - Cleanroom Process Overview* of this series of Cleanroom Handbooks. This handbook, Volume 4, describes the activities of the Specification Team for each cycle of project development. The specification activity may include the preparation of both system and software specifications depending on the needs and status of the project. Specifications define the requirements placed on the system so it can satisfy its mission and the behavior the system must exhibit to fulfill the assigned requirements. A *system specification* defines requirements and behavior for a solution which integrates some combination of hardware, software, communications, human behavior, etc. A *software specification* defines requirements and behavior for the software components of the system. The focus of this volume is on the preparation of software specifications. The techniques discussed are useful for preparing system specifications but in order to keep the handbook within a manageable focus, this presentation is limited to software.

1.1 Background and Motivation

The fundamental objective of software specification is to describe requirements in **black boxes.** The black box is a pure form for defining system requirements without jumping into design decisions prematurely. The major tasks involved in preparing a specification are

> <u>Analysis of the problem domain to gain sufficient understanding to support the creative inventions required to develop the software solution.</u> This analysis activity recognizes that it is a rare event when a software solution does not replace some existing solution and it is always important to understand the system being replaced in order to invent a replacement. The major activities performed to understand the problem domain include information collection, reverse engineering, black box modeling, and Markov usage profile modeling. In many (maybe even most) situations, the first goal of problem domain analysis is to develop a black box description of the behavior of some aspect of the current system. In every case, the final goal is to have a black box description of the behavior of the system to be implemented. [Section 3]

Analysis of the solution domain to gain sufficient understanding to support the creative inventions required to develop the software solution. This analysis recognizes that the software is to be implemented in some environment and one or more aspects of that environment must be better understood in order to determine how it will influence the behavior of the software solution. The major activities performed to understand the solution domain include information collection, reuse analysis to search for existing components that can be included in the final design, black box modeling to determine the precise behavior of objects that will play a part in the final system solution, and developing prototypes to investigate some aspect of a possible solution. [Section 4]

The creative steps of inventing the visible behavior of the software solution that is to be developed. The visible behavior of a software solution is defined by the stimuli and responses that are invented to communicate with the software and the behavior that is desired in terms of what responses the software should produce as a result of all the stimuli that the software has received previously. The goal of the creative step is invent the most cost effective software to meet the mission and to record the details of the invented stimuli and responses and the invented behavior in terms of a black box function. [Section 5]

The recording of the results of the analysis and the inventive steps in the specification volumes. Specifications are written documents that define the software to be implemented. [Section 5]

Maintaining intellectual control of the specifications during software development and certification process so the final specification fully matches the software as implemented and certified. The major activities performed include making minor revisions to the specifications to make them clearer or to develop solutions to minor discrepancies found during development and certification, configuration management and answering questions as they arise to insure full understanding. [Sections 6 and 7]

Revising the specifications. In some cases the development or certification activity will uncover a problem that needs to be studied or some new fact about the world with which the software interacts is uncovered or changed. The result is that the specifications no longer reflect what is needed. In this case the specifications must be updated to reflect new or revised inventions. [Section 7]

Determining specifications is the most critical, yet least understood, phase of systems development. Many system and software development projects fail because of inadequate understanding of problem requirements. In many cases, even when a system is completed, it does not solve the target problem. It has also been observed that the identification and correction of errors in specifications consume a major portion of system development time and resources. Specification tasks entail close cooperation between the Cleanroom Specification Team, the customer and system users. Thus, behavioral skills are just as important as technical skills in order to get the system specification 'right'. It is imperative that developers have solid methodological support for this critical phase of system development.

As befits its importance, many methods and techniques have been developed to perform system and software specification. A survey of the most well-known methods is found in "A Comparison of Techniques of External System Behavior" by Alan Davis in September 1988 **Communications of ACM.** We observe that current specification methods exhibit several well-known problems which the use of box-structures techniques associated with Cleanroom are designed to solve. The specification problems which the box-structure techniques solve include:

- The elicitation of system objectives and requirements from customers and other system stakeholders is a difficult process. Communication skills among developers, customers, and domain experts are essential. Methods (e.g.; Joint Application Design - JAD) and tools (e.g.; group decision support systems - GDSS) have been devised to support requirements elicitation and gathering. However, many communication obstacles inhibit the collection of accurate information. The principal obstacle is finding a convenient way to define actual or desired behavior in a form that is free of implementation complexities so all interested parties can reason about the intended behaviors.

- Reverse engineering techniques are not used to advantage in system specification. Typically, one or more systems already exist that perform some of the desired system's required functions. Making effective use of the existing systems by reverse engineering them is an important step of specification.

- A majority of methods use the same representation for requirements modeling and specification. It is very difficult, however, for one representation to serve both as a communication interface with the customer and as a formal statement of requirements suitable for rigorous analysis and communication with developers. Graphic forms are appropriate for communication models while more formal languages are appropriate for specifications.

- Specification analysis lacks an established set of metrics to evaluate the 'goodness' of the requirements. There are needs for both quantitative and qualitative standards to evaluate specification consistency, completeness, clarity, etc. The advent of more formal specification languages provides better ways of defining and measuring requirement metrics.

- Most specification methods do not support an integrated system development process. True process integration requires common underlying concepts throughout the complete system development. The final system specification should be based on the same concepts and representations as are used in subsequent phases of system design and implementation.

- Current methods fail to recognize the iterative nature of the systems development process. It is foolish to think that complete system requirements can be frozen at the beginning of system development. Controlled, incremental system development is a more realistic and practical paradigm.

1.2 Cleanroom Specification Process Model

As described in *Volume 1* of the Cleanroom Process Manuals, Cleanroom system and software development can be described in a detailed process model. A complete development project for system S can be divided into cycles for specification.

Process P4, Specification Preparation, results in the preparation of the desired specification. This process assumes the entire specification activity is subdivided into cycles. Each cycle is short relative to the entire specification effort. The major idea behind this division into cycles is to help keep the specification effort under intellectual control. Each cycle has three main activities as follows:

1. Prepare a plan for the cycle based on current progress and appraisal of results.

2. Execute the plan for the specified period.

3. Appraise progress and the direction the design is heading.

The appraisal can either result in a continuation of current direction, a modest change in direction, or a major change in direction which results in a replanning of the project. In modest sized projects a cycle of one or two weeks is sufficient. In major projects it may be desirable to permit a cycle as long as 4 weeks. It is very rarely if ever desirable to permit cycles on more than 4 or 5 weeks. That is the way projects get out of control and the result is wasted money. The detailed process model for P4 is:

```
proc P4: Specification Preparation
    [This process results in specifications for the software to produced in this spiral (project).]
    [P4: Specification Preparation]
    while
        (project spiral not terminated and plan still has more cycles to be completed) or not
        (M4.i.5.2 management decision was specification suitable to initiate development)
        [M4.i.5.2 results in one of the following decisions: Specifications ready, another
        specification cycle is necessary, stop specifications and replan.]
    do
        [P4.i handles the investigations for (including prototyping), preparation of, and appraisal
        of the specifications, resulting in the specifications for a project/spiral when all cycles are
        complete.]
        run  P4.i: Software Specification for Cycle i; [where i is the next cycle in the sequence
             i= 1...i...m according to the project plan]
        if  M4.i.5.2 management decision was specification problems-replan project or (this is
            last cycle in plan and not M4.i.5.2 management decision was specifications suitable
            to initiate development)
            [M4.i.5.2 results in one of the following decisions: Specifications ready, another
            specification cycle is necessary, stop specifications and replan.]
        then
            [PCS1: is a common service procedure which results in an updated Software
            Development Plan or a decision to terminate current project/spiral.]
            run  PCS1: Replan project/spiral;
            if  MCS1.4 management decision indicates need to terminate project spiral
            then
                terminate current spiral
            fi;
        fi;
    od;
corp;
```

1.3.1 Process For Specification Cycle i

The engineering task for the specification of cycle i is labeled P4.i, Cleanroom Specification for Cycle i. The process is expanded into engineering subprocesses (i.e., tasks) for execution by Cleanroom-trained software engineers. The detailed process model for P4.i is:

```
proc P4.i: Software Specification for Cycle i
    [P4 handles the investigations for (including prototyping), preparation of, and appraisal of
    the specifications, resulting in the specifications for a project/spiral.]
    do [P4.i: Software Specification for Cycle i]
        run P4.i.1:   Prepare Plan for Cycle i;
        for
            Allocated time period
        do
            con
                [P4.i.2 studies the problem domain to guide system design]
                run P4.i.2:   Problem Domain Analysis-Cycle i;
                [P4.i.3 studies the solution domain to guide system design including searching for
                reuse opportunities]
                run P4.i.3:   Solution Domain Analysis-Cycle i;
                [P4.i.4 is a complete preparation cycle through the six volume specification]
                run P4.i.4:   Prepare Cycle i Specifications;
            noc;
        od;
        run P4.i.5:   Appraise Cycle i Specifications;
    od;
corp;
```

A brief summary of each of the five specification tasks is presented in the balance of this section. Sections 2 through 5 describe the specification processes (P4.i.2 - P4.i.5) in greater detail.

1.3.2 Prepare Plan for Cycle i

The first task of specification cycle is to prepare a plan for cycle i.

```
proc P4.i.1:   Prepare Plan for Cycle i
    do [P4.i.1:   Prepare Plan for Cycle i]
        con
            M4.i.1.1: Establish Objectives For Cycle i;
            M4.i.1.2: Allocate Time Period To Cycle i;
            M4.i.1.3: Prepare Plan For Cycle i;
        noc;
    until
        Completion Conditions achieved for combination of M4.i.1.1, M4.i.1.2 and M4.i.1.3
    od;
corp;
```

Planning in the Cleanroom environment is discussed in *Volume 3 - Cleanroom Planning*. Issues unique to planning the specification cycle are discussed in Section 2.

1.3.3 Problem Domain Analysis for Cycle i

The Specification Team needs to understand the problem domain in sufficient detail to invent and specify a new solution. The formal way we gather understanding and then act on that understanding to develop a solution is called analysis. Typically a great deal of the effort associated with the analysis of the problem domain is fact finding and then analysis to develop a useful model for communications, for reasoning, and, eventually, for recording in specification document. In most situations, much of the analysis relates to understanding the current system and then determining how the systems should be changed. The major activities performed to understand the problem domain include information collection, reverse engineering, black box modeling, and Markov usage modeling. In many (maybe even most) situations the first goal of problem domain analysis is to develop a black box description of the behavior of some aspect of the current system. In every case the final goal is to have a black box description of the behavior of the system to be implemented.

A detailed model for P4.i.2 Problem Domain Analysis-Cycle i is:

```
proc P4.i.2:  Problem Domain Analysis-Cycle i
    do    [P4.i.2:   Problem Domain Analysis-Cycle i]
        con
            for each analysis method in plan
            do
                case [select the appropriate analysis method]
                    analysis method is
                part  Information Collection [collect and document required information]
                    S4.i.2.1:  Information Collection;
                part Reverse Engineering [develop black box function for each system]
                    S4.i.2.2:  Assemble Information for System(s)
                    for    each system in need of reverse engineering
                    do
                        case
                            existing system type is
                        part Manual
                            S4.i.2.3:  Manual System Reverse Engineering;
                        part Automated
                            S4.i.2.4:  Automated System Reverse Engineering;
                        part Hybrid
                            S4.i.2.5:  Hybrid System Reverse Engineering;
                        esac;
                    od;
                part Black Box Models   [develop black box function model for system to be
                                        built, document in Specification volumes 2 and 3]
                    S4.i.2.6:  Develop and Analyze Black Box Model;
                part Markov Usage Models  [develop Markov usage model for system to be
                                          built, document in Specification volume 5]
                    S4.i.2.7:  Develop and Analyze Markov Usage Model;
                esac;
            od;
        noc;
    until
        sufficient Completion Conditions achieved for all selected analysis methods and (time
        period up or sufficient understanding has been obtained)
    od;
corp;
```

Section 3 of this manual discusses engineering practices that are useful in accomplishing each of these 7 activities in 4 sections as follows:

3.1 Information Collection
3.2 Reverse Engineering
3.3 Developing and Analyzing Black Box Models
3.4 Developing and Analyzing Markov Usage Models

1.3.4 Solution Domain Analysis-Cycle i

In addition to fact finding about and analyzing the problem domain it is often necessary to analyze the solution domain in order to gain sufficient understanding of possibilities to understand the best direction to guide inventions. This analysis recognizes that the software is to be implemented in some environment and one or more aspects of that environment must be better understood to determine how it will influence the behavior of the software solution. The major activities performed to understand the solution domain include information collection, reuse analysis, black box modeling, and developing prototypes to investigate some aspect of a possible solution.

A detailed model for P4.i.3 Problem Domain Analysis-Cycle i is:

```
proc P4.i.3:  Solution Domain Analysis-Cycle i
    do    [P4.i.3:    Solution Domain Analysis-Cycle i]
        con
            for each analysis method in plan
            do
                case [select the appropriate analysis method]
                    analysis method is
                part Information Collection [collect and document required information]
                    S4.i.3.1:  Information Collection;
                part Reuse Analysis    [consider reuse options effect on system behavior]
                    do
                        S4.i.3.2:  Browse Reuse Repositories To Find Match For Needs;
                        S4.i.3.3:  Prepare Cost/Benefit Analysis;
                        S4.i.3.4:  Select Potential Reuse Modules for Integration into System;
                    until
                        Completion Conditions achieved for combination of S4.i.3.2-S4.i.3.4
                    od;
                part Black Box Models   [document unspecified objects as black box fox
                                         functions]
                    S4.i.3.5:  Determine Black Box Behavior Of Other Solution Domain Objects;
                part Prototyping    [design, develop, conduct and appraise prototype experiments
                                     to appraise solution possibilities]
                    do
                        S4.i.3.6:  Develop prototype software using Cleanroom practices;
                        S4.i.3.7:  Conduct and appraise prototype experiment
                        S4.i.3.8:  Store prototype components in project reuse repository;
                    until
                        Completion Conditions achieved for combination of S4.i.3.7 and S4.i.3.8
                    od;
                esac;
            od;
        noc;
    until
        sufficient completion Conditions achieved for all selected analysis methods and (time
        period up or sufficient understanding has been obtained)
    od;
corp;
```

Section 4 of this Manual discusses engineering practices that are useful in accomplishing each of these 8 activities in 4 subsections as follows:

4.1 Information Collection
4.2 Reuse Analysis
4.3 Determine Black Box Functions For Objects Known To Be In Solution Domain
4.4 Prototyping

1.3.5 Prepare Cycle i Specifications

The Specification Team develops understanding of the problem and solution domains and begins to invent the solution for the new software to be developed. They record their understanding and the inventions in the specification volumes. They develop the writings in stages, providing more and more details until they have the specifications complete. In early cycles the emphasis is on the early volumes and in latter cycles the emphasis switches to the latter volumes.

A detailed model for P4.i.4 Prepare Cycle i Specifications is:

```
proc P4.i.4:  Prepare Cycle i Specifications
  do  [P4.i.4:  Prepare Cycle i Specifications]
    con
      S4.i.4.1:   Record Cycle i Results In Mission Volume;
      S4.i.4.2:   Record Cycle i Results In User's Reference Manual Volume;
      S4.i.4.3:   Record Cycle i Results In Black Box Function Volume;
      S4.i.4.4:   Record Cycle i Results In Mission Validation Volume;
      S4.i.4.5:   Record Cycle i Results In Usage Profile Volume;
      S4.i.4.6:   Record Cycle i Results In Construction Plan Volume;
    noc;
  until
    Completion Conditions achieved for S4.i.4.1 through S4.i.4.6
  od;
corp;
```

The results of the software specification activities are published as a software system specification in six volumes. The six volumes are:

Volume I -	The Mission
Volume II -	User's Reference Manual

Volume III -	Black Box Functions
Volume IV -	Black Box Verification
Volume V -	Usage Profile
Volume VI -	Construction Plan

The Mission Volume defines the mission the software is to perform in terms of function and performance. The User's Reference Manual Volume describes the user transactions at appropriate levels of abstraction. The Black Box Functions Volume defines the precise stimulus-response behavior required of the software system. The Black Box response to every stimulus is based on the preceding stimuli sequence and the real time performance required. The Black Box Verification Volume presents the justification that the Black Box specifications found in Volume III are correct as written. The Usage Profile Volume defines the statistical usage profile of the users external interaction with the software. The Construction Plan Volume defines the sequence of increments in which the software is to be developed and certified.

The software specification tasks are discussed in Section 5 of this process manual.

1.3.6 Appraise Cycle i Specifications

A management review and evaluation of the cycle i specifications are essential before software system development begins.

```
proc P4.i.5:  Appraise Cycle i Specifications
  do    [P4.i.5:  Appraise Cycle i Specifications]
    M4.i.5.1:  Review and Evaluate Cycle i specifications;
    M4.i.5.2:  Management Decision: (1) Specifications Suitable To Initiate Development
               or (2) Specification Problems-Replan Project or (3) Continue Specification
               Effort with cycle i+1;
  od;
  until
    Completion Conditions achieved for M4.i.5.2
corp;
```

Planning in the Cleanroom environment is discussed in *Volume 3 - Cleanroom Planning*. Issues unique to planning the specification cycle are discussed in Section 6.

SECTION 2: PLANNING THE SPECIFICATION CYCLE

Experience indicates that short specification cycles that encourage frequent, short reviews of progress and directions are effective for developing quality specifications. The important aspects in planning for the specification cycle are to assess the current situation, develop objectives that can be accomplished in two to four weeks, and assess progress and accomplishments. In this way the Specification Team can organize to have good interactions with all stakeholders in the system.

The specification cycle planning process is:

```
proc P4.i.1:  Prepare Plan for Cycle i
  do [P4.i.1: Prepare Plan for Cycle i]
    con
      M4.i.1.1:  Establish Objectives For Cycle i;
      M4.i.1.2:  Allocate Time Period To Cycle i;
      M4.i.1.3:  Prepare Plan For Cycle i;
    noc;
  until
    Completion Conditions achieved for combination of M4.i.1.1, M4.i.1.2 and M4.i.1.3
  od;
corp;
```

Planning in the Cleanroom environment is discussed in *Volume 3 - Cleanroom Planning*. Which levels of management that will participate in the planning and decision making will depend on the circumstance. In most situations the Specification Team Leader and the Software Engineering Manager will make all decisions. In other circumstances they will want to get others involved. The important issue is that there be good coordination between the planning activity P4.i+1.1 and the appraisal activity P4.i.5. In planning for cycle i+1 the Specification Team Leader and the Software Engineering Manager need to take into account all the findings made during the appraisal process. It is the appraisal step that needs good representation from all interested parties to the specification.

Specification Teams are faced with many situations. When using Cleanroom practices they have a firm idea of where they need to get to in terms of the documents they need to prepare and the inventions they need to make. The starting point is always different depending on the situation that exists when the project starts. Plans for specification cycles must accommodate the starting situation and be adjusted dynamically as new aspects of the problem and solution domain are uncovered. In the following paragraphs, guidance is provided on what problem and solution domain techniques can be usefully applied depending on the situation.

Specification Teams must be prepared to handle many different starting points for systems development. The specification process may be faced with situations that vary from no current system, manual or automated, in place to a fully functional system that simply needs to be ported to new hardware or software. Situations may also exist where several existing systems perform various required functions of the new, desired system. It is a critical specification task to evaluate the potential of these existing systems to provide important guidance and transformation or reuse opportunities for system development. In this section, a framework is presented for analyzing different starting points for Cleanroom specification. Guidance is provided on how to adapt the specification process appropriately.

The Specification Team evaluates the 'starting point' for development of the new system. Information on existing systems is gathered and analyzed via reverse engineering methods. Four principal development options exist for the required components of the new system. The complete system specification will consist of component specifications based on one or more of the following options:

- *Creation* - No reuse or reengineering opportunities exist for system components. The full Cleanroom processes of specification, development, and certification are applied to the creation of these system components from scratch.

- *Reuse* - Existing system components are discovered that can be reused in the new system. Minimal modification is needed to integrate these components into the system specification. Effective reuse requires very little specification and development work. Cleanroom certification will be performed as the reused components are integrated into appropriate development increments and tested. In cases where the project is being performed in the MegaProgramming environment then the application development project has been preceded by a Domain Analysis and the implementation of assets that will be residing in a reuse repository. The Specification Team will need to determine how best to reuse these domain assets.

- *Transformation* - Existing system components are found to provide the required functionality, however, some form of transformation is needed to integrate the components into the new system environment. As examples: a manual system can be automated, an automated system can be moved to a new hardware platform, or an automated system can be reprogrammed in a new programming language. The role of the Specification Team in component transformation is to clearly specify the target system environment and define the transformation tasks on the existing components. The Development Team will perform the transformations and the component will undergo certification testing as it is integrated into the appropriate system increment.

- *Reengineering* - An existing system component may provide an excellent base on which to build a component for the new system. The existing component must incorporate new requirements and/or new functional enhancements to fulfill its mission. The reengineering of system components involves understanding the current system behavior (i.e., reverse engineering) and modifying the behavior to enhance and improve the system. The Specification Team modifies the specification of the recovered component to include new requirements and enhancements. This specification is integrated into the overall system specification delivered to the Development Team. The system component is developed and certified as a newly created component.

A system specification may begin with varying starting points. However, with a thorough analysis of specification options, the Specification Team can identify the most effective methods to apply to their given situation. The following Table I presents several common specification starting points and how the Cleanroom process is applied.

Table I - Specification Starting Points

Situation	Specification Options
Existing software functionality to be duplicated on another platform in another language with maybe a more modern solution architecture. No functional enhancements are to be made. Two cases: (a) have source code (b) no source code (e.g., MBC case with no source code)	- Reverse engineer existing software to understand functionality. - Specify needed transformation tasks to transform existing system components to new architecture and/or new language.
Existing software functionality to be duplicated on another platform in another language. Minimal functional enhancements are expected.	- Reverse engineer existing software to understand functionality. - Reengineer current specification to include new functional enhancements. - Specify needed transformation tasks to transform existing system components to new language.
Existing software to be extended with new requirements. Two cases: (a) no new components (b) new components (e.g., COFT)	- Reverse engineer existing software to understand functionality. - Reengineer existing component specifications to include new requirements. - Create new component specifications with new requirements.

Existing software needs to be understood before further planning and specification can proceed.	- Reverse engineer existing software to understand functionality.
Existing manual process to be automated and improved.	- Reverse engineer existing manual process to understand functionality. - Reengineer existing component specifications to include new requirements. - Create new component specifications with new requirements.
New automated process to be created. No existing systems provide required functionality.	- Create new component specifications to satisfy required functionality.

SECTION 3: PROBLEM DOMAIN ANALYSIS

Specification projects usually begin with a heavy emphasis on understanding the problem domain so the Specification Team gains familiarity with the problem for which a solution is required. However, the Specification Team must never lose sight of their basic objective; that is, describing the system requirements as a black box. Thus, information in the problem domain should be gathered and described as completely as possible in black box formats. In other words, the team should concentrate on understanding the problem requirements in terms of available stimuli, needed responses, and the behaviors that transform the stimulus histories into responses. The process for understanding the problem domain is P4.i.2.

```
proc P4.i.2:  Problem Domain Analysis-Cycle i
  do   [P4.i.2:  Problem Domain Analysis-Cycle i]
    con
      for each analysis method in plan
      do
        case [select the appropriate analysis method]
          analysis method is
        part  Information Collection [collect and document required information]
          S4.i.2.1:   Information Collection;
        part Reverse Engineering   [develop black box function for each system]
          S4.i.2.2:   Assemble Information for System(s)
          for   each system in need of reverse engineering
          do
            case
              existing system type is
            part Manual
              S4.i.2.3:   Manual System Reverse Engineering;
            part Automated
              S4.i.2.4:   Automated System Reverse Engineering;
            part Hybrid
              S4.i.2.5:   Hybrid System Reverse Engineering;
            esac;
          od;
        part Black Box Models  [develop black box function model for system to be built,
                                document in Specification volumes 2 and 3]
          S4.i.2.6:   Develop and Analyze Black Box Model;
        part Markov Usage Models [develop Markov usage model for system to be built,
                                  document in Specification volume 5]
          S4.i.2.7:   Develop and Analyze Markov Usage Model;
        esac;
      od;
```

```
        noc;
    until
        sufficient Completion Conditions achieved for all selected analysis methods and (time
        period up or sufficient understanding has been obtained)
    od;
corp;
```

Each of these activities are discussed in the following sections as follows:

Information Collection	Section 3.1
Reverse Engineering	Section 3.2
Developing and Analyzing Black Box Models	Section 3.3
Developing and Analyzing Markov Usage Models	Section 3.4

Specification activities support the understanding, gathering, modeling, and analysis of system requirements in the form of *black boxes*. It is important to recognize the high-level system view of a solution first. A system solution is typically composed of hardware, software (newly developed and packaged), and human behavior; as well as effective interfaces among components. An integrated system solution is developed to make best use of these components.

The following sections provide guidance for developing a specification.

3.1 Information Collection

A necessary task of the specification team is to understand the domain of the system application. The customer environment and perspective must guide all specification activities. Well-honed communication skills are needed on the specification team to elicit requirements information from customers, managers, and domain experts. It is important to spend more ^Xxme in specification making sure system requirements are well understood so that costly re-specifications and re-designs are avoided. While enabling methods (e.g., brainstorming, Joint Application Development) and automated support (e.g., The Requirements Apprentice from MIT) have been proposed, the most effective way to understand the problem domain is to work closely with the customer and put out the effort needed to fully comprehend the customer's needs.

The sources of information are many and varied. The primary sources are the managers, users, and operators of the potential system. If current systems are in use, then documentation such as system manuals, user manuals, system logs, and current application programs may be useful sources of information. Gathering business process information from individuals (managers, users, operators) requires good interpersonal communication skills. The two primary techniques are interviews and questionnaires.

In information systems development, the purpose of interviewing prospective system managers and users is to make explicit the information processing procedures, needs, and objectives of the business process. The current information processing system will be some combination of people and machines, and the planned system, some new combination of people and machines, all of which exhibit black box, state box, and clear box behavior. Both the existing and planned systems will require explicit box structure descriptions. Thus, the knowledge gained in the interviewing process must eventually be represented in terms of box structures.

The interviewing process is intended to reveal box structure behavior to the interviewer. Often these box structures will emerge in fragmentary form, and will require corroboration and elaboration through additional interviews and feedback sessions. For this reason, interviews should not be regarded as solitary and stereotyped events, but rather as a continual process of discussions with individuals and groups to arrive at common understandings and objectives. Early box structure definitions, while often incomplete, can nevertheless be used to advantage in these discussions, as a means to focus on the correctness and completeness of planned system behavior.

The questions asked during a interview should focus on box structure behavior, but the words and phrases employed need not depend on box structure terminology. Even though a person being interviewed has no knowledge of box structure techniques, it is still possible to discuss box behavior in very precise terms. Consider, for example, the following questions, phrased in the everyday language of business, and their interpretation in terms of box structure concepts:

Question 1: "Do previous transactions against a credit account affect the processing of a current transaction?"

Box Structure Interpretation: "What is the black box behavior of credit account transaction processing?"

Question 2: "What information must credit account transaction processing have on hand in order to process a current transaction?"

Box Structure Interpretation: "What state information must be retained in the credit account processing state box?"

Question 3: "How does credit account processing combine the information it has on hand with a current transaction to produce output and update the information on hand"?

Box Structure Interpretation: "What are the transactions of the credit account processing state box?"

Question 4: "What steps are required to process a transaction in credit account processing?"

Box Structure Interpretation: "What is the clear box behavior of credit account processing?"

Question 5: "Is account verification performed before, during, or after credit account processing?"

Box Structure Interpretation: "How are account verification and credit account processing related in terms of a box structure hierarchy?"

Interviews are an effective way to gather accurate and timely information. Questions can be asked and responses clarified on the spot. Participants in an interview feel actively involved in system development. However, interviews are time-consuming for both the developers and the persons interviewed, so the number of interviews should be kept to a minimum and each interview should be short and to the point.

The effectiveness of an interview is directly proportional to the preparation for it. The following guidelines support effective preparation:

1. Define the purpose of the interview. Fishing expeditions rarely result in quality information. Know the system area within which the person interviewed is expert and stick to that area.

2. Only interview selected individuals. It is not necessary to interview everyone and time is limited. Start with upper management and work down the organization hierarchy. This provides the developers with an idea of how the system fits into the overall organization.

3. Be prepared with specific questions for each interview. Do homework and be ready to guide the direction of the interview.

4. Schedule the interview at the interviewee's convenience. However, set a strict deadline for all interviews to be completed. If an individual is not available in that time frame, schedule an acceptable substitute.

An interview has three phases, namely an *opening*, a *body*, and a *closing*. In the opening, state the purpose of the interview, establish the legitimacy of the interview, and achieve a rapport with the person interviewed. In the body, move from general, open questions to specific, detailed questions. Vague answers should be clarified. Take clear notes or ask to record the discussion. The notes should be summarized after the interview. The closing should leave both parties satisfied with the interview. If questions remain, schedule another interview. Offer to send a copy of the interview summary to the person interviewed. Prompt feedback allows any misunderstandings to be discovered and rectified.

Questionnaires are used when information is needed from a large group of individuals, usually system users. The design of the questionnaire is very important. The questionnaire must be clearly written and easy to complete or it will be ignored by most users. It should be checked for clarity and misunderstandings as carefully as a computer program, and test cases run to verify its effectiveness. Analysis of the completed questionnaires is facilitated by the use of computers to tally and statistically analyze responses.

As an understanding of the system problem is evolving, the Specification Team should begin the creative process of proposing and evaluating potential solutions. Alternate solution strategies are developed, analyzed, discarded, modified, and adopted. Understanding the solution domain depends on the specification team's experience and expertise with system hardware, software, telecommunications, human-computer interfaces, human capabilities, and other areas required for a system solution.

The true skill of a Specification Team lies in its ability to understand the opportunities and constraints found in the problem domain and to apply the most effective solution approaches. The creative development of the required system extends from system specification through system development and certification.

3.2 Reverse Engineering

Very frequently automation projects begin with the objective of improving how some current organizational process or processes are performed. The objective of the automation may be to reduce cost, improve performance, correct problems, adopt the current process(s) to a new or changing environment, or improve the process to provide^Xx new service that offers a competitive advantage.

No matter what the reasons for the automation project the starting point in such situations is the current system(s). Frequently the understanding of and documentation of these systems is not exactly what is required. The ideal form to document behavior about a system is as a black box function that describes the external behavior of each affected system. Since most automation efforts are replacing or augmenting systems that were implemented before the development of black box functions was common practice, the desired black box function is usually not available. That means the information must be extracted from the information that is available. This process is known as reverse engineering since what we are doing is looking at artifacts, often the code itself, to extract system knowledge that can be used to reason about the system.

The process for reverse engineering of existing systems for cycle i is included in process P4.i.2. The portion of that process that deals with reverse engineering is extracted below.

```
.
.
part Reverse Engineering
  S4.i.2.2:   Assemble Information for System(s)
  for  each system in need of reverse engineering
  do
    case
      existing system type
    part Manual
      S4.i.2.3:   Manual System Reverse Engineering;
    part Automated
      S4.i.2.4:   Automated System Reverse Engineering;
    part Hybrid
      S4.i.2.5:   Hybrid System Reverse Engineering;
    esac;
.
.
.
```

A brief overview of reverse engineering concepts is presented, followed by a description of the engineering tasks in the process.

3.2.1 Reverse Engineering of Systems

Reverse engineering is the process of analyzing an existing system for the purpose of:

- identifying system components and their interrelationships,
- understanding the functionality of the system components, and
- representing the system components in abstract forms for reengineering the current system and reuse in new system developments.

Reverse engineering is used for many important purposes. Design recovery provides for the recapture of the system design from the system implementation. Redocumentation of an existing system records the system's functions in usable documentation. Reverse engineering sets the stage for the reengineering of systems to add new system enhancements and to improve system maintenance, understandability, and performance.

The ideal way to reverse engineer a system is to use box structures. Existing systems are found in natural forms such as program code, standard operating procedures, flow charts, etc. These natural forms are converted into clear box representations via transformation techniques. Once in a structured clear box format, box structure derivations can be performed to produce functional abstractions of the system in state box and black box formats. Black box functions are the target of the reverse engineering process.

3.2.2 Identify Existing Systems

An initial activity in Cleanroom specification is to discover and gather all pertinent information on the requirements for the desired system. Principal sources of this information are the existing systems currently used to satisfy the needed functions. Only in a very few cases is the desired system functionality not being met by some form of system either manual or automated. Of course, the new system is expected to be much improved over the existing system in terms of additional functionality, enhanced performance, more user-friendliness, greater security/integrity, or some combination of these factors.

It is essential that the Cleanroom Specification Team identify pertinent existing systems and understand their functionality as the starting point of the systems development project. Existing systems can be classified as follows:

- *A manual system* - A manual system built around human processes currently handles the required functions.

- *An automated system* - A fully automated system is implemented in computer hardware and software.

- *A hybrid system* - A hybrid system combines manual and automated components.

Based upon the identification of existing systems, the next steps involve understanding these systems via reverse engineering.

3.2.3 Reverse Engineer the Existing Systems

For each existing system, the Specification Team applies reverse engineering methods to understand the functionality of the system and to recover design and requirements information. The amount of time and effort devoted to reverse engineering a given system is based on the team leader's evaluation of how important the existing system is for the specification of the new system. The extent of the reverse engineering tasks is also greatly influenced by the quality and quantity of the available resources on the existing system. Types of available system resources include:

- *Development Information*: If the system is automated, information should be available on its original development. Such information would include requirements specifications, analysis and design documents, and test data. The specification team must also be aware of the development process and methods used by the original development team.

- *User and System Documentation*: An effective system, whether manual, automated, or a hybrid, must include some form of documentation. User documentation will explain the procedures for using the system functions. System documentation describes the internal workings of the system for effective maintenance and operations.

- *Program Code*: Automated systems contain program code. In most cases, the source code is available for analysis. However, some teams may only have access to object code for the existing system. This requires more sophisticated reverse engineering techniques for abstracting the functionality of object code.

- *Standard Operating Procedures (SOPs)*: For manual and hybrid systems there typically exist SOPs for the performance of the systems. SOPs are presented in the form of flow charts, algorithms, decision tables, or other procedural representations.

- *Customers and Users*: Interviews with the customers and users of the existing systems will provide valuable information on the effectiveness of the existing systems. In particular, information on the human-computer interfaces can be elicited.

- *Operators and Managers*: The operators and managers of the existing systems will give a different perspective than the customers and users. Information on performance, integrity, security, accuracy, reliability, and other quality issues can be gained by selected interviews.

The system information gained from the above sources provides the basis for reverse engineering the existing systems. In some cases the available information may be sufficient to answer all questions.

Box structure analysis can be applied to derive a black box system representation. The goal is to describe an existing system in box structure representations to support further rigorous analysis and reverse engineering. System understanding is achieved by describing the system as a usage hierarchy of referentially transparent clear boxes. Methods for transforming natural procedures and structured program code into clear box formats are presented in Mills, Linger, and Hevner, **Principles of Information Systems Analysis and Design**, 1986 and Linger, Mills, and Witt, **Structured Programming: Theory and Practice**, 1979.

The analysis continues by building an increasingly abstract description of an existing system in a recursive, bottom-up fashion. Detailed clear box descriptions of subsystems are derived to state box and black box representations. These subsystems are then represented as black boxes within procedural clear boxes at the next higher level of system description. This process continues until the complete system is described and understood at the top-level black box behavior.

The Box Structure Analysis Algorithm is:

Box Structure Analysis Algorithm

```
I. Identify System Components

     Task 1 - Modularize System into Components
     Task 2 - Represent System Components as Clear Boxes

II. Abstract Box Structure Behavior

  for Each Clear Box
     do
        Task 3 - Derive State Box
        Task 4 - Derive Black Box
        Task 5 - Analyze Black Box Behavior
     od

III. Build Higher Level System Abstractions

  if Top-level Black Box achieved
  then
     Algorithm completed
  else
     con
        Task 6 - Group Related Black Box Components
        Task 7 - Define Clear Boxes at Higher Abstraction Level
     noc
  fi

  Repeat Step III until all black boxes are used in at least one clear box.

IV. Build Box Structure Usage Hierarchy

  Task 8 - Build Box Structure Usage Hierarchy with Clear Boxes

Repeat Algorithm beginning with Step II
```

End of Box Structure Analysis Algorithm

The eight tasks of the Box Structure Analysis Algorithm are briefly described:

Task 1 - Modularize System into Components

The Specification Team investigates the current representation of the existing system; from information written procedures to actual software code. The system is divided into cohesive modules; with each module containing a single function of the system. The modules should be small enough to understand without any further decomposition. This is the first tasks of bottom-up analysis.

Task 2 - Represent System Components as Clear Boxes

Each module must be transformed from its current representation to a clear box representation. This transformation is straightforward for automated systems with program code. For manual systems with modules in natural language or flow chart formats, a more involved, but still straightforward, transformation may be necessary.

Task 3 - Derive State Box

Once the module is in a clear box format, a derivation is performed to discover the state box representation. The clear box to state box derivation removes all procedural details and integrates all internal black boxes into a single non-procedural state box behavior.

Task 4 - Derive Black Box

From the state box, the black box representation of the module is derived. The state box to black box derivation removes all state designs. The state information is preserved in the stimulus history of the black box. The black box behavior is a design-free representation of the behavior of the system.

Task 5 - Analyze Black Box Behavior

The black box behavior of the current module is analyzed for a true understanding of the behavior of the module. This behavior can be compared to the intended behavior to verify consistency.

Task 6 - Group Related Black Box Components

As part of the bottom-up construction of the box-structured system usage hierarchy, black box components at the current level are grouped into an abstraction at the next higher level. The purpose of this step is to identify the appropriate black boxes that interact to define a coherent function within the system. This grouping is guided by the information contained in the existing system.

Task 7 - Define Clear Boxes at Higher Abstraction Level

New clear boxes are defined for the functions at the next level of the box structure usage hierarchy. The information flow among the internal black boxes identified in Task 6 is defined.

Task 8 - Build Box Structure Usage Hierarchy with Clear Boxes

All clear boxes defined in Task 7 describe a sub-hierarchy of box structures. These sub-hierarchies are represented and stored as part of the box structure analysis of the existing system. Each iteration through this algorithm builds higher and higher levels of abstraction until, finally, a single box structure usage hierarchy representations the complete description of the system.

For each new clear box defined in Task 7, the algorithm is repeated beginning with Step II.

In the reverse engineering process, information is gained on each existing system. Then, depending on whether the system is manual (Task S4.i.2.3), automated (Task S4.i.2.4), or a hybrid of the two (Task S4.i.2.5), an appropriate level of reverse engineering is performed in order to understand the system's functionality and applicability to the specification of the new system. The principal differences among these three tasks is the form and quality of the information the Specification Team must work with and the manner in which the Box Structure Analysis is adapted to the existing systems. Manual systems require considerable work in step 1 of the Box Structure Analysis algorithm to form the natural procedures into box structure formats. Automated systems typically have some forms of design documentation already available; maybe even in electronic format. Hybrid systems will require an integration of techniques to handle manual system and automated system representations.

Once the reverse engineering of the existing system is complete, the black box behavior of the system is available for any necessary redesign. The Specification Team's role is to present this design opportunity to the Development Team in the new system's specification.

The analysis tasks will produce reusable information on the existing systems that were analyzed. This information will be placed into a **project reuse repository** for traceability and use by the Development Team. The process of reengineering an existing system by box structure analysis and box structure redesign is illustrated in Figure 1. For many system developments, this reengineering process is sufficient to development the new system directly from the starting point of an existing system.

Examples of performing box structure analysis of existing systems can be found in **Principles of Information Systems Analysis and Design**. The Navy Supply Case Study in Chapter 1 is an excellent example of an automated system whose analysis provided a deeper understanding of the system's true functionality. Based on this understanding, the system was completely

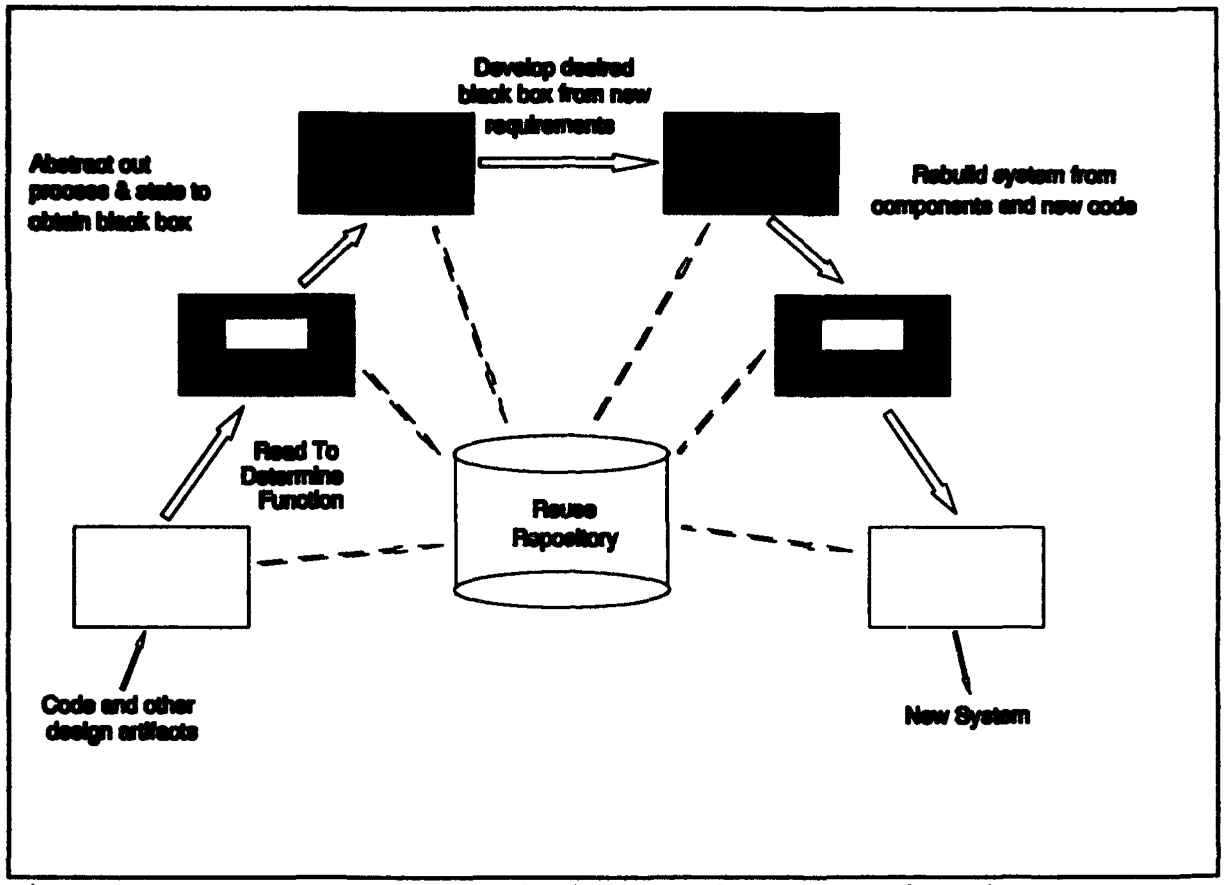
Develop desired black box from new requirements
Abstract out process & state to obtain black box
Rebuild system from components and new code
Read To Determine Function
Reuse Repository
Code and other design artifacts
New System

Figure 3.2.1: Box Structure Analysis for Software Reengineering

redesigned to provide improved quality and performance. Chapter 5 of the text contains two analysis examples of manual systems. A tax schedule is analyzed to its black box representation and redesigned to be more effective. A business' hiring process, presented as a flow chart, is analyzed via box structures. An effective technique for transforming the flow chart into a clear box is presented. The process analysis points out several errors in the original process and suggests improvements for the redesign of the hiring process.

3.3 Develop Black Box Models

Once the raw specification information is gathered from the customer (Task S4.i.2.1) and from the reverse engineering of existing systems (Tasks S4.i.2.2 - S4.i.2.5), the Specification Team is prepared to creatively design the system specification in terms of black box transactions. In this section, techniques are presented for structuring the specification information into black boxes and hierarchies of black boxes. Such black box models will provide effective brainstorming opportunities for the Specification Team as well as provide valuable communication interfaces with the customer.

The black box is a 'pure', design-free representation of desired system behavior. External stimuli enter the black box and responses are returned to the external environment. A meaningful black box system has a well-defined behavior for mapping sets of stimuli into sets of responses. No internal details of system state or procedurality are described in a black box.

During requirements determination, black box structures are used to describe required system behavior. Black box requirements are based on sets and functions that can be described in mathematical notation for appropriate systems or subsystems or in well-structured natural language in a given context. In any case, a black box is defined by a mathematical function from histories of stimuli to the next response. This detailed, low-level behavior is termed a system *transition*. Let S be the set of possible stimuli, and R be the set of possible responses of a system or subsystem. The black box transition function, say f, will map historical sequences of such stimuli, in this case S*, to responses, R, shown in the form, f: $S^* \rightarrow R$.

The description of the transition function f will be very complex for any reasonably complex system with large numbers of possible stimuli and responses. But this is a complexity of the system that must be recognized and addressed. Getting this complexity under control early in specification and design is much better than letting it go and trying to fix it later when the software doesn't work well.

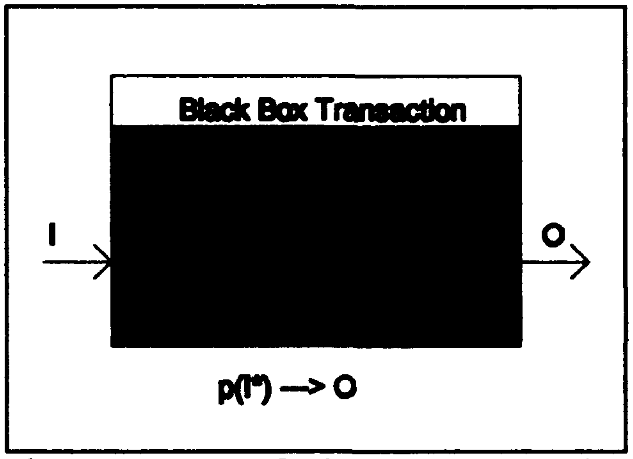

Figure 3.3.1: Black Box Transaction Graphic

In order to manage this complexity, higher levels of system abstraction are needed to describe required system behavior as system *transactions*. A black box transaction is a pattern of black box transitions in which all responses, but the last, are predictable by the user. The last response is new information. The sequence of stimuli to the transaction is called an *input* and the sequence of responses, including the last response, is called an *output*. In the same manner as a black box transition, the black box transaction is defined formally as a function, say p, from a history of inputs, I*, to an output, O, in the form, p: $I^* \rightarrow O$. The black box transaction is shown in Figure 2.

In essence, the transition describes low-level, system and software-oriented behavior while the transaction describes higher level behavior that is better understood by humans (e.g., customers, users, and developers). While detailed transitions are needed for eventual system software design and implementation, system transactions are presented as requirements at higher levels of abstraction. In box-structured requirements determination, requirements are elicited in terms of transactions, i.e., functions of inputs to outputs. Thus, while the system behavior will be designed and implemented in terms of thousands of individual system software transitions, system requirements are typically described with less than one hundred transactions.

One observation is that a typical system will support many kinds of users; many of whom are there to make the system run for the others. For example, an on-line, all-day system will be started every day by operating people, it will need people who tune its performance, people who build the database, people who train other people in its use, and so on, in addition to the principal users to add data, retrieve data, etc. So many kinds of transitions and transactions will be called for every day. All these transactions need to be identified and planned for from the very beginning, not brought in as after thoughts to those of the principal users.

The input into the requirements gathering phase is some form of problem statement, typically presented as an English document. Requirements gathering tasks are performed in order to collect all information that will help to determine the particular requirements of a system that solves the presented problem. The goal is to format this information into black box transactions. To support this goal, a simple requirements gathering method consisting of three steps is shown:

3.3.1 Requirements Gathering Procedure

Step 1: Identify Inputs - A list of system inputs is generated via information gathering tasks.. It should be recognized that these inputs will be at various levels of abstraction, from databases and files to simple data variables and physical signals (e.g., a clock pulse). All potential and available inputs should be listed. An analysis of the necessity and sufficiency of the inputs will be performed later. The list of inputs is:

$$\bar{I} = (I_1, I_2, \ldots, I_i).$$

Step 2: Identify Outputs - A list of required outputs from the system is generated. Close interaction with the customer and users is needed to develop a complete output list. The output list is defined as:

$$\bar{O} = (O_1, O_2, \ldots, O_j).$$

Step 3: Form Black Box Transactions - The black box behavior that relates the input history to the required outputs is described. A set of black box transactions is generated, $(p_1, p_2, \ldots, p_k)$. Each transaction is a function from the input history to a set of required outputs, i.e.,

$$p_m(\bar{I}*) \rightarrow O'_m \text{ where } O'_m \subseteq \bar{O}.$$

All required outputs must be produced by one or more transactions.

End of Requirements Gathering Procedure.

The following template can be used to contain the input list, the output list, and the set of transactions that map input histories into outputs.

Template 1: Black Box Inputs and Outputs for Box <boxname>

List inputs and outputs for the box <boxname>.

inputs

I1:	Invocation
IF2:	Sensor value(sensor reading)
<I3>:	<i3>
<I4>:	<i4>

responses

OF1:	Open file for input
OF2:	Read value
<O3>:	<o3>
<O4>:	<o4>

Design Notes:
OF1 and OF2 are commands sent to the File.

- In developing inputs and outputs it is necessary to adopt the view of the software that will be receiving the inputs and issuing the outputs.

The discovery of inputs, outputs, and black box transactions is an iterative process. The next phase of requirements modeling forms this information into a hierarchical structure of transactions. As this hierarchy expands, further steps of information gathering will be needed to achieve consistency and completeness of the system specification. But with a rigorous framework, the information gathering comes under intellectual control. The black box postpones state and procedure invention, but provides a framework for dealing with the black box of a complex system with many different kinds of users and therefore many different black box inputs. As noted before, the need is to identify the entire behavior required in the black box before going into system design.

3.3.2 Model System Requirements as Transactions

The ability to handle requirements information at various levels of abstraction is essential in order to maintain intellectual control over the system specification process. An abstraction hierarchy of black box transactions is an effective framework for building a model of system requirements. This hierarchy supports both the top-down decomposition of system transactions and the bottom-up composition of transactions into higher level transactions. The identification of reusable subsystems and the recognition of essential common services are supported by this box-structured modeling process.

A template for modeling black box transactions is shown below:

Template 2: Black Box Transactions for Box <boxname>

```
begin black box transaction I* || I: <boxname>

black box sub-transaction I* || I1: Invocation is
   B01.01  OF1: Open file for input;
   B01.02  OF2: Read value;
   XOB;

black box sub-transaction <I2>: <i2> is
      <>
   XOB;

end black box function I* || I: <boxname>
```

Design Notes:

- Given the inputs and outputs identified in Template 1 it is straightforward to write down each of the outputs in terms of input histories.
- In general, the steps of discovering inputs, outputs, and behaviors are performed concurrently. One first develops some inputs and outputs and then begins to refine the Black Box transaction. This leads to identification to the need for more inputs and/or outputs. This cyclic process continues until it seems the definition is complete. Then one moves on to examine and to verify the transaction. These tasks may uncover the need for more inputs and/or outputs or modifications to the control structure for the subtransactions.

The transaction hierarchy, shown in Figure 3, is constructed by modeling requirements information in meaningful transactions at various levels of abstraction. Typical model development would begin by identifying the top-level (i.e., level 1) system transactions. These transactions would be grouped to encompass the functional requirements for the complete system. Then, using the process of stepwise refinement, each transaction can be decomposed into a group of sub-transactions at the next level of the hierarchy. At each step of refinement, the group of transactions at the next level are verified for consistency with the parent transaction and are analyzed for transaction closure.

In parallel with the top-down decomposition of required transactions, an analysis of the bottom-up composition of detailed requirements into higher level abstractions can be performed. This analysis is especially critical when reusable components from libraries or existing systems are available. The modeling of the transaction hierarchy becomes a challenging, iterative process of

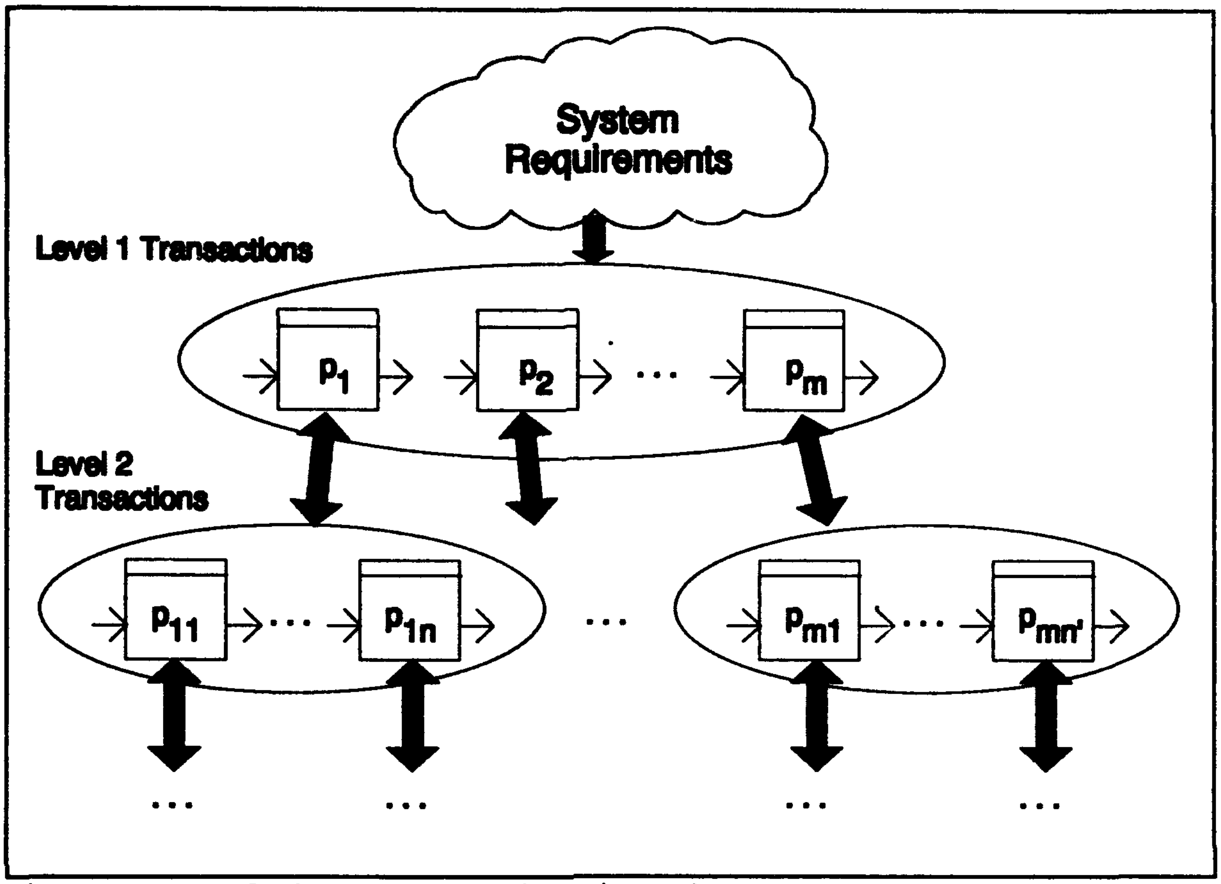

Figure 3.3.2: Black Box Transaction Hierarchy

matching customer and business needs with the resources available to satisfy those needs.

The transaction hierarchy is built through many iterations of requirements gathering, modeling, and analysis of the model. This graphic representation of the system requirements is an excellent communication device for interaction with customers, users, and business managers. Template 3 provides a means to record the brainstorming and analysis associated with building the transaction hierarchy.

Template 3: Transaction Hierarchy

<Graphical representation of the transaction hierarchy (e.g., Figure 3.3.2) is maintained here.>

Design Notes:

- A graphic editor is used to provide a visual representation of the transaction hierarchy. This format allows brainstorming to occur on issues of transaction decomposition and transaction composition. Transactions can be moved up, down, and throughout the hierarchy until an appropriate specification is determined.

- Each transaction in the hierarchy references its BDL description in Template 2.

Black box transactions are 'pure' representations of functional system requirements. Any additional information, such as detailed information flows, control flows, or data structures, constrains the freedom of the system designer to produce the most effective system design. However, such constraints may be valid based on the need to integrate with existing systems or human behavior. Thus, the transaction hierarchy can be extended to include three types of additional information:

- State Constraints
- Procedural Constraints
- Non-Functional Requirements

Often system requirements do contain design constraints on such things as the availability and use of data or the need to conform to a defined procedure. The operating or management systems environment within which a system will be embedded may provide opportunities or place demands on the specification and design of the desired system. If a specification can be modified to make more of the software reusable with equal power, that should be done. In addition, certain 'non-functional' requirements, such as performance, behavioral, and documentation standards, can be stated in structured English forms, or in system performance models (e.g., Petri-nets). It is important during specification reviews that the system owners understand that any non-functional requirements beyond a black box are constraints upon the system's design freedom. In this way, many non-essential requirements can be discovered and eliminated.

Information on constraints and non-functional requirements can be represented with appropriate models or structured English statements. These artifacts are linked to the affected transactions in the hierarchy. For example, state constraints could be described by Entity-Relationship Diagrams, procedural constraints could be described by control flow charts, and non-functional requirements could be described by performance models.

Once the transaction hierarchy is accepted by the customer as a true reflection of system requirements, then the requirements are described in a more formal requirements modeling language as a specification. The modeling representations found in Templates 1-3 for transactions and transitions can be stated more precisely as functions in a Box Description Language. This detailed requirements specification appears in Volumes II and III of the Specification Volumes (see Section 5). The representation of requirements in a formal language provides two important advantages:

- Rigorous analysis procedures can be performed on the requirements specification, and
- A consistent, complete, and clear requirements specification is given to the Cleanroom development team. No ambiguities or unnecessary design constraints hamper the creative tasks of system design.

An extended Box Description Language is presented in Appendix A of this volume. Rigorous languages for requirements specification are quite recent. The formality of programming languages is necessary to make assemblers and compilers possible. In a similar manner, formal specification languages bring the definition of system requirements under intellectual control. It may even be possible to execute specifications for analysis purposes.

3.3.3 Analyze Model

Throughout the previous phases of system requirements gathering and modeling, analysis procedures are applied to measure and evaluate the quality of the system specification. Three critical aspects of specification quality can be identified.

Consistency - The rule of consistency is that each group of black box transactions in the transaction hierarchy must be consistent with its higher level parent transaction. In other words, the individual behavior of the transactions must collectively match the behavior defined in the parent transaction. It is important to note that the interactions of the children transactions are yet to be designed. Thus, the transaction hierarchy does not exhibit referential transparency. The lack of referential transparency precludes a formal verification of consistency as can be performed during box structure system design. However, an informal analysis of consistency throughout the transaction hierarchy is essential.

Completeness - Completeness is validated by determining that the problem description is consistent with the black box. This is done to ensure that the black box represents the same function as the original problem.

Closure - Closure can be validated by ensuring that every black box transaction has necessary and sufficient sets of inputs and outputs. This is termed transaction closure. A straightforward algorithm for checking transaction closure in black boxes is presented here. Note that black box transition terminology (i.e., stimuli, responses, functions) is used instead of the transaction terminology (i.e., inputs, outputs, transactions). The mapping between the two terminologies is straightforward. Closure analysis can be applied to both transactions and more detailed transition behaviors in the hierarchy.

Black Box Closure Algorithm

Given:

$S = (s_1, s_2, . . , s_n)$:	complete set of stimuli entering the system
$R = (r_1, r_2, . . , r_m)$:	complete set of responses generated by the system
$F = (f_1, f_2, . . , f_p)$:	complete set of transition functions describing the behavior of the black box

Step 1: Check that all responses are generated:

For all r_j in R there exists a subset S_A of S and a f_k in F such that $f_k(S_A)$ --> r_j.

Step 2: Check that all stimuli are used:

For all s_i in S, there exists a subset S_A of S where s_i is an element of S_A, and there exists a r_j in R and a f_k in F, such that $f_k(S_A)$ --> r_j and $f_k(S_A - s_i)$ -/-> r_j.

Step 3: Check that all functions are used:

For all f_k in F there exists a r_j in R and a subset S_A of S such that $f_k(S_A)$ --> r_j.

End of Black Box Closure.

During this procedure, unnecessary stimuli (inputs) can be deleted and additional needed stimuli (inputs) can be identified and gathered. When requirements are compiled informally by several people, both consistency and completeness problems can arise. A single requirements statement assembled by several people under formal discipline of box structures can better insure both consistency and completeness.

Clarity - Two forms of clarity are needed for effective requirements determination. The requirements model must present requirements in a form understandable to the customer and system users. The system specification must present requirements in a form appropriate for system developers. The box-structured approach provides the flexibility for system requirements to be stated in the language of the problem domain. Effective use of structured English statements at high levels of the transaction hierarchy allows customers and users to better understand the requirements model. The formal, mathematics-based framework of the Box Description Language specification is a clear starting point for detailed system design, with no unnecessary design constraints.

Template 4 can be used to record these analyses along with other analyses performed on the black box definition.

Template 4: Transaction Analysis

List all analyses performed for the transaction hierarchy.

Hypothesis: The transaction groupings in the transaction hierarchy are consistent with parent transactions.

- Analysis Process:

- Results:

- Transaction Modifications:

Hypothesis: The transaction hierarchy contains a complete description of system requirements.

- Analysis Process:
  (1) Make a mapping between each transaction and a section of the problem description.
  (2) Ensure that all parts of the problem description have been covered.

- Results:

- Black Box Modifications:

Hypothesis: Black box closure exists for all transactions.

- Analysis Process:

- Results:

- Black Box Modifications:

Hypothesis: Each transaction is clearly described in terms understandable in the application domain.

- Analysis Process:

- Results:

- Black Box Modifications:

Additional Analyses as Required

3.4 Develop Markov Usage Models

Since the concept of developing Markov usage models is at the basis of certification for Cleanroom, it will be discussed in detail in Section 2 of Volume 6. However, the development of the Markov usage models is the responsibility of the Specification Team.

SECTION 4: SOLUTION DOMAIN ANALYSIS

In addition to understanding the problem domain the specification team needs to understand the constraints and opportunities presented by the solution domain. They need to evaluate solution options by conducting prototype experiments and preparing trade studies. They frequently need to evaluate reuse opportunities so they can reduce development expense by preparing their inventions to take advantage of components that can be acquired commercially, found in one of the many commercial reuse repositories that are coming into existence or in the organizations own reuse repository.

The process for understanding the solution domain is:

```
proc P4.i.3:  Solution Domain Analysis-Cycle i
  do   [P4.i.3:  Solution Domain Analysis-Cycle i]
    con
      for each analysis method in plan
      do
        case [select the appropriate analysis method]
          analysis method is
        part Information Collection [collect and document required information]
          S4.i.3.1:   Information Collection;
        part Reuse Analysis   [consider reuse options effect on system behavior]
          do
            S4.i.3.2:   Browse Reuse Repositories To Find Match For Needs;
            S4.i.3.3:   Prepare Cost/Benefit Analysis;
            S4.i.3.4:   Select Potential Reuse Modules for Integration into System;
          until
            Completion Conditions achieved for combination of S4.i.3.2-S4.i.3.4
          od;
        part Black Box Models   [document unspecified objects as black box fox functions]
          S4.i.3.5:   Determine Black Box Behavior Of Other Solution Domain Objects;
        part Prototyping     [design, develop, conduct and appraise prototype experiments to
                              appraise solution possibilities]
          do
            S4.i.3.6:   Develop prototype software using Cleanroom practices;
            S4.i.3.7:   Conduct and appraise prototype experiment
            S4.i.3.8:   Store prototype components in project reuse repository;
          until
            Completion Conditions achieved for combination of S4.i.3.7 and S4.i.3.8
          od;
        esac;
      od;
```

```
        noc;
    until
        sufficient Completion Conditions achieved for all selected analysis methods and (time
        period up or sufficient understanding has been obtained)
    od;
corp;
```

Each of the four main activities are discussed in the following subsections as follows:

Information Collection	Section 4.1
Reuse Analysis	Section 4.2
Determine Black Box Behavior of Components Known to be in Solution Domain	Section 4.3
Prototyping	Section 4.4

4.1 Information Collection

The Specification Team gathers information on the solution domain using similar techniques as discussed in Section 3. Interviews, questionnaires, literature surveys, vendor surveys, demonstrations, and other means of gathering information are used. Solution domain information sources would include the following:

Organizational Information System Architectures

The organization in which the system will operate may have well-defined IS architecture plans and strategies in place. The system solution must comply with the standards set by the IS architecture. It is important to discover and understand all applicable organizational IS architectures.

Reuse Libraries

The Specification Team must gather information on all available reuse libraries. Information would include content, organization, browsing methods, access methods, and cost of reusing modules from the library.

Benchmarking Studies

Benchmarking is a total quality management technique wherein an organization compares its processes with other organizations to discover the best-in-class processes. Building a new system based on out-dated processes is foolish. It pays to discover the best processes for implementation in the new system.

Existing Systems that Solve the Similar Problems

Similar systems may exist in other parts of the organization or in other organizations. If the team has access to these systems, they can learn good ideas as well as ideas that did not work as well.

Theoretical Results

The Specification Team must stay abreast of the state-of-the-art in the application domain of the system. Customer input to the team will be valuable on domain issues. The newest theories and practices in the domain should be implemented in the system.

Consulting Experts

Application experts and technical experts are often necessary to provide complete and up-to-date information on solution opportunities. For example, the specification of a modern application system may require expertise in multi-media, human-computer interfaces, object-oriented database systems, neural networks, or many other state-of-the-art technologies.

The solution domain information is used by the Specification Team to discover the most effective ways to build the required system.

4.2 Reuse Analysis

The reuse analysis tasks for cycle i are included in process P4.i.3. The portion of that process that deals with reuse analysis is extracted below.

```
.
.
.
part Reuse Analysis   [consider reuse options effect on system behavior]
  do
    S4.i.3.2:   Browse Reuse Repositories To Find Match For Needs;
    S4.i.3.3:   Prepare Cost/Benefit Analysis;
    S4.i.3.4:   Select Potential Reuse Modules for Integration into System;
  until
    Completion Conditions achieved for combination of S4.i.3.2-S4.i.3.4
  od;
.
.
.
```

Major gains in development productivity and system quality can be achieved by emphasizing reuse as early in the system life cycle as possible. As each system and software module is identified during specification, reuse opportunities should be explored in all available reuse repositories. In performing this reuse analysis the Specification Team must be careful to define external behavior but to not limit the internal inventions that are the responsibility of the Development Team.

The three subtasks of reuse analysis are performed sequentially. Each task is discussed in the following three subsections.

4.2.1 Browse Reuse Repositories

During system and software specification, black box transactions and transitions are described in hierarchical structures. Black box definitions provide sufficient information to support the matching of system and software requirements with reusable system and software modules stored in repositories. Access to existing reuse repositories, both within and outside the organization, is needed to support effective reuse.

In box structure terminology, reusable modules within and among systems are known as *common services*. A common service is a data abstraction that is described in a separate box structure usage hierarchy, and used in other box-structured systems. System parts with multiple uses should be defined as common services for reusability. Also, predefined common services, such as database management systems and input/output interfaces, should be used to advantage throughout the box structured system. Box structures directly support the identification and reuse of common services within and among systems.

The Specification Team must have an efficient, user-friendly interface for browsing reuse repositories for requirements matching. Issues of module organization, indexing, requirements representation, and access capabilities are areas of on-going research and development and are not covered in this handbook. Essentially, the browsing subtask allows the specification team to quickly identify reuse possibilities. Such reuse opportunities are then subjected to a thorough cost/benefit analysis.

4.2.2 Analyze Reuse Costs and Benefits

A black box requirement can be matched with existing modules stored for reuse in a repository. During system and software specification the benefits and costs of module reuse and modification can be studied. Another effective use of reusable modules comes during the construction of the transaction hierarchy. Knowledge of existing reusable modules can influence the team's invention of black boxes at the next level in the transaction hierarchy.

As an example of requirements matching, assume there exists a reusable software module with black box transaction behavior, $r(I'^*) \rightarrow O'$, where I' and O' are the inputs to and outputs from the module. Given a black box transaction somewhere in the hierarchy, $p_i(I''^*) \rightarrow O''$, the potential is evaluated for the reusable module to match the transaction requirements. Requirements matching must be done on inputs (I' and I"), outputs (O' and O"), and behavior (r and p_i). If an exact match is not found, several alternatives can be studied:

1. Use the reusable module as is and modify the system requirement to accommodate its behavior.

2. Modify the behavior of the reusable module to match the system requirement.

3. Modify both the behavior of the reusable module and the system requirement in order to produce an effective match.

4. Do not use the reusable module and search for other reuse opportunities or decide to develop a module from scratch to satisfy the system requirement.

A detailed matching algorithm is needed, along with a cost tradeoff procedure to evaluate the most effective reuse strategy.

Opportunities also exist during the modeling of the transaction hierarchy to discover required system common services. A common service is a portion of the system that can be reused in several places in the transaction hierarchy. Reusable modules are common services. The discovery and effective placement of common services in the specification model provides important design and implementation efficiencies in later stages of system development. New common services can also be defined and implemented as reusable modules for future system developments.

4.2.3 Select Reusable Modules and Integrate into Specification

Based on the above analyses, reusable system and software modules are selected, retrieved from the repository, and integrated into the specification. The selection of a reusable module will typically require the modification of interacting transactions to accommodate module input and output interfaces. Thus, the system and software specifications will be defined to fully integrate the reusable modules.

While the effort of browsing reuse repositories, analyzing reuse cost/benefit, and integrating reusable modules in the specification may seem high, the major benefits are accrued during the Cleanroom development and certification tasks. A complete reusable module would include the black box requirement, the state box and clear box designs, the certified software code, and all related documentation. All of these reusable artifacts are integrated into the software system design and implementation.

4.3 Determining Black Box Behavior For Objects Known To Be In Solution Domain

There are many situations where the software object under development will interact with other objects to form a complete solution. Typical situations include projects where functionality is to be added to the software or where the software is to use some hardware devise that is under going parallel development. In such situations the specifiers and the developers need to know how to interact with these other objects.

The best course of action is to apply the Box Structure Analysis Algorithm, as described in Section 3.2.3, to the interface components of the other systems. This provides documentation of the understanding of the behavior of the other object as a black box function. This can be thought of as selected reverse engineering.

4.4 Prototyping

The prototyping tasks for cycle i are included in process P4.i.3. The portion of that process that deals with reuse analysis is extracted below.

```
.
.
.
part Prototyping   [design, develop, conduct and appraise prototype experiments to
                   understand and/or appraise solution possibilities]
    do
        S4.i.3.6:  Develop prototype software using Cleanroom practices;
        S4.i.3.7:  Conduct and appraise prototype experiment
        S4.i.3.8:  Store prototype components in project reuse repository;
    until
        Completion Conditions achieved for combination of S4.i.3.7 and S4.i.3.8
    od;
.
.
.
```

These activities are performed whenever the need for prototyping is discovered during system and software specification. Prototyping is an experimental process of building a well-defined portion of a system in order to discover new information. This information is used to better understand system and software requirements and/or solutions. Prototyping is an effective means of evaluating and resolving risk during system specification and development.

The three tasks of the prototyping process are performed sequentially. These tasks are discussed in the following three subsections.

4.4.1 Develop Prototype Using Cleanroom Process

The decision to develop a prototype is based on a cost/benefit study. This study compares the cost of building the prototype versus the benefit of the information gained. A prototype must have a well-defined objective and must be a self-contained development project. It is easy to get carried away on building a prototype long after the desired information has been obtained.

Examples of typical prototypes include:

User interfaces -	The prototyping of screen interfaces, menu systems, report formats, etc. is performed to obtain information on user behavior patterns and system usage patterns.
Packaged software -	Required system functions are prototyped on commercial software packages to evaluate completeness and efficiency. Different software packages can be compared in this way.
Innovative procedures -	New, innovative procedures for providing system functionality are tested by implementing them in a prototype that contains just enough system support to fully evaluate them. By building a prototype system environment, several alternative procedures can be compared and evaluated.

When a decision is made to prototype a portion of the system, the prototype development process will take on a life of its own. Following the standard Cleanroom development process, as detailed in Volume 1, the prototype will go through stages of planning, specification, development, and certification. A new stage of *experimentation* is then performed. The prototype experiment is designed to provide the information required to satisfy the objective of the prototype.

While the responsibility for the prototype belongs to the Specification Team, the actual specification, development, and certification of the prototype system may be shared among all Cleanroom teams. Several prototype construction scenarios can be considered:

Scenario 1: For small-scale prototypes and rapid prototyping, the Specification Team can perform all specification, development, and certification tasks. This requires that the members of the Specification Team be trained to perform all development and certification tasks.

Scenario 2: For small- to medium-scale prototypes, the prototype is specified, developed, and certified by selected members of the Specification Team who will become members of the Development Team and Certification Team as the project progresses. This scenario is likely in Cleanroom organizations where projects are completed sequentially with little overlap.

Scenario 3: For medium- to large-scale prototypes, the prototype specification is completed by the Specification Team and given to the Development and Certification Teams for prototype development and certification. This scenario occurs when the Specification Team has a large specification to work on and the other Cleanroom teams are available to support the specification tasks before they begin their work on the actual system.

The selection of the most beneficial scenario for executing the scenario will be based on factors such as prototype size, required resources, availability of Cleanroom teams, number of concurrent projects, time constraints, etc. Upon completion of the prototype, the experiments are again the responsibility of the Specification Team to execute. Experimental results are gathered and analyzed, leading to an appraisal of the prototype.

4.4.2 Conduct and Appraise Prototype Experiment

The objective of a prototype is to gain new information in the Cleanroom specification process. After construction of the prototype, one or more experiments are performed on the prototype to gather the information desired. An appraisal of the experimental results can lead to one of the following management decisions:

i) The information gained from the experiment satisfies the objectives of the prototype.

ii) The information is incomplete. New experiments are designed to gather more information.

iii) The information is incomplete. The prototype development plan is revised in order to expand or modify the scope of the prototype. The Cleanroom development process for the prototype is revised to accommodate the revisions. Additional experiments are designed for the revised prototype.

iv) The information is incomplete and a reanalysis of the cost/benefit of the prototype precludes further expansion of the prototype or the experiments. The information gained is used to the extent possible to satisfy the prototype objectives.

The completion criteria for this task is determined by the management decisions in i) or iv) above.

4.4.3 Store Prototype Modules in Reuse Repository

Regardless of the appraisal of the prototype experiment, the software modules developed and certified within the prototype should be placed in the project reuse repository. The Cleanroom development team can use these modules as a starting point for the development of the actual software system modules. Thus, the development and certification efforts on the prototype will be conserved during the development and certification of the final system. The identification of potential reusable modules was discussed in Section 4.2.

SECTION 5: PREPARING THE SPECIFICATION

Process P4.i.4, Prepare Cycle i Specifications, consists of six tasks performed concurrently and interactively.

```
proc P4.i.4:  Prepare Cycle i Specifications
  do   [P4.i.4:  Prepare Cycle i Specifications]
    con
      S4.i.4.1:   Record Cycle i Results In Mission Volume;
      S4.i.4.2:   Record Cycle i Results In User's Reference Manual Volume;
      S4.i.4.3:   Record Cycle i Results In Black Box Function Volume;
      S4.i.4.4:   Record Cycle i Results In Mission Validation Volume;
      S4.i.4.5:   Record Cycle i Results In Usage Profile Volume;
      S4.i.4.6:   Record Cycle i Results In Construction Plan Volume;
    noc;
  until
    Completion Conditions achieved for S4.i.4.1 through S4.i.4.6
  od;
corp;
```

The concurrent tasks of problem and solution domain analysis support the information gathering and analysis needed to produce the detailed software specifications of the designated system transactions. The results of the software specifications are formed into six specification volumes. The six specific tasks of Cleanroom software engineering are described in terms of building these six volumes. The completion criteria for the six tasks are the completion of these six volumes.

5.1 Volume I - The Mission

The purposes of the Mission Volume are (1) to define the mission that the software is to fulfill, (2) to define the context in which the software will be operating, and (3) to record the argument that the software satisfies the defined mission. The following template provides a typical outline of Volume I.

Software Specification For [Project Name]

Volume I - The Mission

Section 1: The Mission Statement

The mission statement identifies the purpose and objectives of the software to the customer and the customer's organization. The mission statement specifies what effects the software is intended to have on all stakeholders. A mission statement should rarely be more than one page. It should be a clear, concise statement of intent. A stakeholder is any noncompetitive individual or organization directly affected by the software's behavior and performance; for example, stockholders (it is a rare software system that does not affect profit), customers, suppliers, distributors, employees, government agencies, etc.

Section 2: The Detailed Mission

This section defines the requirements allocated to the software in order for the software to be in a position to satisfy the mission statement. This section should define each requirement for the software in a concise statement; typically, one or two sentences.

Section 3: The System Context

This section defines the context or the environment in which the software will be operating. The results of the system specification tasks will provide most of this information. The precise contents of this section will differ depending on the situation. The following lists topics that may be included:

- A description of the current, automated or manual system to be improved by the planned development.
- The hardware and software available for use.
- Existing systems with which the new system must interface.
- The resource (e.g., personnel, budget, facilities) environment planned for the new system.

The exact contents will depend on what is required to insure that the contemplated role of the software in the environment to be created is clear.

Section 4: The Mission Validation Argument

This section contains the arguments that the software is implemented and accomplishes the defined mission, the software will accomplish its intended purpose in the environment in which it will be operating.

5.2 Volume II - User's Reference Manual

The purpose of the User's Reference Manual is to provide a user's view of the software's functionality. The use of transactions and the transaction hierarchy gives a structured picture of the functions in the system. The system context is also described along with a description of the desired system performance. The following template provides a typical outline of Volume II.

Software Specification For [Project Name]

Volume II - User's Reference Manual

Section 1: The Environment

The software exists in a larger environment. This section defines the environment that the software will operate in. Typical subsections are:

- Hardware
- Peripherals
- Interface devices
- Communications networks
- Operating systems
- Software applications
- Databases
- Personnel (e.g., operating staff, users, managers)

Section 2: Software Transactions

Transactions are defined as functions in terms of input-output behavior. Lists of transactions are determined based on a thorough understanding of the problem domain. Classes of transactions would include:

- User transactions
- Administrator transactions
- Manager transactions
- Start-up/Shut-down transactions

- Error transactions

Transaction hierarchies are developed to support both top-down and bottom-up requirements determination.

With each transaction, clear designation of inputs and outputs is required. These inputs and outputs must be formally related to the black box stimuli and responses as detailed in Volume III.

Section 3: Software Interface Specifications

Precise syntax describes the invocation and the activation of the software components. The stimuli and response interfaces for each transaction are specified in this section.

Section 4: System Performance

In many systems there are performance constraints associated with the transformation of inputs into outputs. These requirements are typically related to response times, throughputs, and accuracy (i.e., precision) of the transaction. All performance standards must be defined.

Section 5: Undesired Events

Software is designed to handle undesired events as well as the desired or planned events. In this section, all undesired events are enumerated and the response that will be taken to each undesired event is defined along with the corrective actions the user may take in those cases where corrective action is required.

Section 6: Software Initiation

This section describes what to do when the software arrives from the development team. It describes how the software is to be installed, configured, how files are to be initialized, how the software is invoked for the first time, how to handle failures, and all other activities associated with making the software ready to use.

Section 7: Glossary

List of words used in the User's Manual.

Section 8: Index

Location of words in the User's Manual.

5.3 Volume III - Black Box Function

The purpose of the Black Box Functions Volume is to define an implementation-free view of the system and software being designed. This implementation-free view is obtained by specifying functions which define the conditions in terms of stimuli histories that cause the presentation of each possible response that can be produced by the software. The following template provides a typical outline of Volume III.

Software Specification For [Project Name]

Volume III - Black Box Function

Section 1: The Stimuli

This section lists all stimuli and for each stimulus provides an exact name, a brief description, and a reference name or number used in the black box function for identification purposes. The precise syntax and format of each stimulus is required. Also required is the method by which the stimulus is obtained from the system environment. Transaction closure must ensure that all stimuli are listed for the desired system. This will require several iterations to develop a complete stimulus list.

Section 2: The Responses

This section lists all the responses and for each response provides an exact name, a brief description, and a reference name or number used in the black box function for identification purposes. The precise syntax and format of each response is required. Also required is the method by which the response is presented to the system environment. Transaction closure must ensure that all responses are listed for the desired system. This will require several iterations to develop a complete response list.

Section 3: The Black Box Functions

This section contains a sub-section for each black box function in the software system. A black box hierarchy is based on the transaction hierarchy of Volume II. New functions will be needed to completely specify the transaction requirements.

For each black box, the required behavior is defined by a function from stimulus histories to responses. The black box functions can be expressed in terms of pseudo-code or program tables.

It is very important that these functions be defined carefully and are well written so that everyone who is interested in the software can read and comprehend them. It is very important that no state data or other implementation-dependent inventions creep into the definition of the black box functions because if they do, they are no longer black box functions. Once they become something besides black box functions, they lose their independence and ease of comprehension.

5.4 Volume IV - Black Box Validation

The Black Box Validation presents an argument which justifies the correctness of the Black Box Functions. This argument is not easy to make, but serves as the basis for accepting or rejecting the system specification. The following template provides a typical outline of Volume IV.

Software Specification For [Project Name]

Volume IV - Black Box Validation

The Black Box Verification Argument

This volume contains the argument that the system is implemented so that it satisfies the defined black box functions then the software will fulfill its intended mission. This argument can be made as formal as necessary to satisfy verification requirements; for example, for safety-critical systems, complete proofs of system correctness may be needed. In any case, it is important that this argument be well presented since it is necessary to satisfy all interested parties that the software accomplishes its mission. Lower level design work should not progress until all stakeholders are satisfied with the black box functions and the verification arguments.

Verification arguments can be built along the lines of the transaction and black box hierarchies developed in Volumes II and III. Grouping functions into transactions allows a natural approach for verifying that complete and closed functional behavior satisfies system requirements.

5.5 Volume V - Usage Profile

The Usage Profile contains the definition of anticipated software use by each class of system user. This definition is required to develop test scenarios in accordance with usage specifications so it is possible to make measurements of the reliability of the software in its intended environment.

The following template provides a typical outline of Volume V.

Software Specification For [Project Name]

Volume V - Usage Profile

The construction of the system's usage profile using Markov chains is described in detail in *Volume 6 - Certification Team Practices*. The sections of this volume are:

Section 1: Usage States

In this section, the usage state definition is established and the individual usage states are enumerated. Some argument about the validity and completeness of the definition is also included.

Section 2: Stimuli and Stimuli Distributions

This section contains an enumeration of the stimuli that appear in the system. Additionally, the distributions of the stimuli in various modes of operation are also presented, to provide an accurate model of the expected use of the software to be developed.

Section 3: Usage Profile

This section contains the stimuli and the transitions among the usage states caused by applying each stimulus. This information can be organized as a state transition diagram (STD) with the usage states comprising the state set and the arcs connecting the states labeled with stimuli. An equivalent formulation is a transition matrix with the states as indices and the stimuli as the matrix entries.

Once the arcs are established, each is assigned a probability of occurrence. These probabilities are documented in this section along with any data or analysis that was used in obtaining the probabilities.

Section 4: Assumptions

All the assumptions made in defining the usage states or establishing the stimuli distributions should be organized and included in this section for convenient review by interested stakeholders.

Section 5: Usage Profile Analysis

The statistical analysis of the usage profile should be included in this section. The data can be included in its entirety but should also be summarized as appropriate for the application. Any specific items of interest to stakeholders should be identified and highlighted.

If several iterations of adjustments to the stimuli distributions are necessary, all such analyses do not have to be included. However, it is helpful to show the specific results that led to successive changes to the distributions and some documentation about how the changes were conceived. This way a history of the development of the usage profile is maintained.

5.6 Volume VI - Construction Plan

The purpose of the Construction Plan is to develop a plan for the Cleanroom development and certification teams during the actual software development. The plan contains the actual modules and functions to be a part of each increment. The result is a list of modules that the development team needs to produce for each increment, and the functionality available for the certification team to certify for each increment. The following template provides a typical outline of Volume VI.

Software Specification For [Project Name]

Volume VI - Construction Plan

Based on a thorough analysis of system requirements and available resources (e.g., budget, time, software engineers), the desired system is divided into self-contained increments for development and certification. Each Increment is defined in a separate section of this volume in the following manner:

Increment j:

1. Specification references for increment j

2. Increment Architecture

   System and software context for increment j
   Black Box Hierarchy for increment j

3. Module 1 - Black Box Function Name

   References to Volumes III and V for black box function and certification information.

4. Module 2 - Black Box Function Name

   References to Volumes III and V for black box function and certification information.

Continued for all software modules included in increment j.

SECTION 6: APPRAISING CYCLE I SPECIFICATIONS

Experience indicates that short specification cycles that encourage frequent, short reviews of progress and directions is effective for developing quality specifications. The important aspects in planning for the specification cycle are to assess the current situation, develop objectives that can be accomplished in two to four weeks and then assess progress and accomplishments. In this way the Specification Team can organize to have good interactions with all stakeholders in the system.

The specification cycle evaluation process is:

```
proc P4.i.5:   Appraise Cycle i Specifications
   do   [P4.i.5:  Appraise Cycle i Specifications]
      M4.i.5.1:  Review and Evaluate Cycle i Specifications;
      M4.i.5.2:  Management Decision: (1) Specifications Suitable To Initiate Development
                 or (2) Specification Problems-Replan Project or (3) Continue Specification
                 Effort with Cycle i+1;
   od;
   until
      Completion Conditions achieved for M4.i.5.2
corp;
```

Planning in the Cleanroom environment is discussed in *Volume 3 - Cleanroom Planning*. Either the Specification Team Leader of the Software Engineering Manager will host the review and the analysis. The normal participants for the reviews and contributors should be specified in the Software Development Plan for the project. The main issue is to get a sufficient number of system stakeholders involved so that the team can be sure that it is solving the right problem. Short, frequent, meaningful reviews are the best way to insure that inventions being made by the Specification Team are the inventions required to solve the real problem. It is important that there be good coordination between the planning activity P4.i+1.1 and the analysis activity P4.i.5. In planning for cycle i+1 the Specification Team Leader and the Software Development Manager need to take into account all the findings made during the analysis process.

SECTION 7: REVISING SPECIFICATIONS

There are two situations where the specifications are involved in maintenance. There are the normal interactions in which the Specification Team interacts with the Development and Certification Teams. In this situation, they must consult with these teams and keep the specifications under configuration management. The process is

```
proc P7: Specification Configuration Management
  [This process results in the specification being kept current with all changes required due to
  normal project findings.]
  do
    con
      S7.1 Consult with Development and Certification Teams About Specification Issues;
      S7.2 Update Specification as required;
      S7.3 Increase Understanding of Problem and Solution Domains;
    noc;
  until
    project spiral terminated and Completion Conditions for combination of S7.1 and S7.2
    achieved
  od;
corp;
```

The second situation reflects the dynamic nature of software systems development. This dynamism precludes the freezing of specifications before beginning the Cleanroom development and certification of software increments. Specifications must be viewed as dynamic and susceptible to change throughout the Cleanroom development process. The overall Cleanroom process model as detailed in *Volume 1* contains decision points in the process where revisions can be made to the specification. Therefore, the revision process is a common service and identified as PCS2, Update Specification.

```
proc PCS2: Update Specifications
  [This process results in an updated specification]
  do [PCS2: Update Specification]
    if Question or Issue or stimulus from outside project or error discovery causes a
       specification change
    then
      do
        con
          SCS2.1:   Increase Understanding of Problem and Solution Domains;
          SCS2.2:   Update Specification;
        noc;
        SCS2.3:   Publish Change Sheets;
        MCS2.4:   Management Decision: (1) OK continue current plan with revised
                  specifications or (2) Revised specifications require replanning
      until
        Completion Conditions for MCS2.4 achieved
      od;
    fi;
  od;
corp;
```

The primary tasks of this process are to increase the Specification Team's understanding of the problem and solution domains and to revise the specification based on this new information. Then appropriate specification changes are published and a management decision is made whether the changes require any project replanning. The following sections describe the tasks of these two processes:

Increase Understanding of Problem and Solution Domains	Section 7.1
Update Specification	Section 7.2
Publish Specification Changes	Section 7.3
Perform Replanning	Section 7.4

7.1 Increase Understanding of Problem and Solution Domains

Many issues and questions arise during all phases of the Cleanroom development process. Most issues and questions come from inadequate understanding of the specifications or the specifications are incomplete. In either case, the Specification Team must provide clarifications and answers expeditiously.

Specifications will evolve over time as system customers, users, managers, and developers add, delete, and modify requirements. While such changes should not be allowed to get out of control, it is normal for requirement changes to occur and they must be effectively managed. The Specification Team is responsible for maintaining configuration control over the specification documents and alerting all Cleanroom project members when changes to the specification are posted.

The first task required of the Specification Team when faced with a issue or question is to fully understand it. This will require talking with the individual(s) who raised the problem. If it is a simple misunderstanding of the specification, the matter can be settled by a clarification and complete answer.

Deeper system issues will necessitate an increased understanding of the problem domain and the solution domain. The Specification Team may need to gather more requirements information through interviews, document reviews, and prototyping. Also, more information on solution opportunities can be gathered. This new information is integrated with the existing information to model revised system and software requirements in the form of black box transactions and transaction hierarchies. The revised requirements are reviewed with customers and the appropriate Development Team members to verify that the issues and questions are satisfied by the proposed revisions. The task of increasing the understanding of system requirements continues until the revisions are verified.

7.2 Revise Specification

Configuration control of the specification documents are critical to the control of the Cleanroom project. Revisions to the specification are entered under strict change control. Two sets of changes are made:

Requirements Models - Changes are made to the appropriate black box transactions and the transaction hierarchy. New analyses on consistency, closure, completeness, and clarity are performed on the altered transactions. New and modified transactions are evaluated for software development.

Requirements Specification - The six volume software specification is revised where appropriate. New analyses of reuse opportunities may be performed.

The specification revisions are evaluated in a team review to verify correctness and how well the changes satisfy the original issue or question. The appropriate customers and Development Team members are also brought in to discuss the impacts of the revisions on the current state of the project.

7.3 Publish Specification Changes

Strict configuration control of the specification documents are essential. Once the specification revision is accepted, a formal specification revision is published and disseminated to all Cleanroom teams. A new revision number may be assigned to the new specification version. Based on the extend of the revision, a complete set of specification documents may be published or just the changed pages of the current documents. Project managers must ensure that the specification documents are updated promptly and no further work is done based on outdated specifications.

7.4 Perform Replanning

It is a management decision whether the changes to the specification necessitates replanning the Cleanroom project. Changes in resource (e.g., time, budget, personnel, equipment) allocation may be needed to develop a system to meet the new specifications. The decision would be either to continue the project with the current plan or to revise the project plan. The issues of Cleanroom planning, including the analysis of replanning, are presented in *Volume 2 - Cleanroom Planning*.

SECTION 8: TAILOR SPECIFICATIONS TO INCREMENT J

In Volume VI, Construction Plan, of the specification, the software is formed into increments, or accumulations of increments, for development and certification. As the Cleanroom Development Team and Certification Team begin their work on *increment j*, the Specification Team must tailor the specification for that increment. This is the first subtask, P5.j.1, of process P5.j.

```
proc P5.j: Software Development and Certification Preparation
  [For each P5.j, the specification is tailored, then the software is designed to the code by the
  Development team (P5.j.2), while the Certification team does the work necessary to prepare
  for certification of the increment (P6.j.3).]
  do   [P5.j: Software and Certification Planning]
    run P5.j.1:   Tailor specification to increment/accumulation j;
    con
      run P5.j.2:   Prepare for Certification of Accumulation j;
      run P5.j.3:   Increment j Development;
    noc;
  until
    Completion Conditions achieved for P5.j.2 and P5.j.3 or P5.j.2 team decision indicates the
    need for replanning or specification revisions
  od;
corp;
```

The tasks of P5.j.1 are:

```
proc P5.j.1:  Tailor specification to increment/accumulation j
  do [P5.j.1: Tailor specification to increment/accumulation j]
    con
      S5.j.1.1:   Tailor Black Box function to increment/accumulation j;
      S5.j.1.2:   Tailor Usage Profile to increment/accumulation j;
    noc;
  until
    Completion Conditions achieved for S5.j.1.1 and S5.j.1.2
  od;
corp;
```

The required tailoring tasks are defined by the increment decisions presented in Volume VI of the specification. Once the decisions on software increments are made, the Specification Team will group the information found in the Black Box Functions (Volume III), and the Usage Profile (Volume V) into a convenient format for presentation to the Development Team and the Certification Team. Tailoring will involve reorganization of these three volumes along with any specification revisions necessary to support the increment definitions. In many situations the

Development Team and the Certification Team will be tasked to do this work. In this way they gain familiarity with the work to be done.

APPENDIX A - BOX DESCRIPTION LANGUAGE BNF

The BNF below is presented to clarify the valid syntax for Black, State and Clear Boxes. More specifically, one will see that a part of the BNF that appears here for the Black, State and Clear boxes also appears in the respective templates for each box.

Notation:
- $^+$ means 1 or more
- * means 0 or more

```
<design object>    ::=   <black box> | <state box> | <clear box>

<black box>     ::=   black box function S* || S <object name> is
                  [black box subfunction S*|| <stimulus name> is
                     <structure>+
                  XOB;]+
                  end black box function S* || S  <object name>

<state box>     ::=   state box function S: <object name> is
                  [state box subfunction <stimulus name> is
                     <structure>+
                  XOB;]+
                  end state box function S:  <object name>

<clear box>     ::=   begin clear box <object name> is
                  [stimuli
                     <stimulus name>*
                   responses
                     <response name>*
                   state
                     <state name>*
                   data variables
                     <variable name>* |
                   var
                     <stimulus name>*
                     <response name>*
                     <state name>*
                     <variable name>*]
                   proc <object name>
                     <CBstructure>+
                   corp;
```

```
<object name>      ::= object name

<stimulus name>      ::= SXXi:   stimulus name | <variable name>

<response name>      ::= RXXi:   response name | <variable name>

<state name>    ::= TXXi:   state name | <variable name>

<variable name>    ::= variable name

<structure>        ::=    <sequence> | <fordo> | <ifthen> | <ifthenelse> | <case> | <whiledo>
                          | <dountil> | <connoc>

<CBstructure>      ::=    <structure> | <run> | <use>

<sequence>      ::=    (<structure> | <statement>;$^+$)$^+$

<fordo>         ::=    for
                  indexlist
               do
                  <sequence>
               od;

<ifthen>        ::=    if
                  <condition>
               then
                  <sequence>
               fi;

<ifthenelse>    ::=    if
                  <condition>
               then
                  <sequence>
               else
                  <sequence>
               fi;

<case>         ::=    case
                  p
               (part(CLi)
                  <sequence>)$^+$
               else
                  <sequence>
               esac;
```

```
<whiledo>       ::=  while
                     <condition>
                do
                     <sequence>
                od;

<dountil>       ::=  do
                     <sequence>
                until
                     <condition>
                od;

<connoc>        ::=  con
                     <sequence>
                noc;

<run>          ::=  run <cs>

<use>          ::=  use <BBstatement>

<statement>     ::=  any statement that is returning a response, presenting a description of
                     actions (non-commentary), or making an assignment.

<condition>     ::=  any logical expression that can evaluate only to true or false.

<cs>           ::=  A name of a common service that is at the current level of the software
                    system's parts hierarchy.

<BBstatement>     ::=  A stimulus to a lower level black box
```